Project Report: Gesture-Controlled Robot For Military Application

Team Members:

- Srestha Sarkar - 1CR23EC211
- Sristi Purkayastha - 1CR23CD064
- Srijan Shivakumar Kakhandaki - 1CR23EC213

Introduction

This project demonstrates the development of a gesture-controlled robot using an Arduino Uno. The robot integrates advanced components like a gyroscope, accelerometers, RF communication modules, and motor drivers to enable intuitive hand-gesture control. The choice of these components was driven by their reliability and compatibility, providing seamless data acquisition and transmission. Designed primarily for military applications, this robot can execute critical tasks such as bomb defusal, border patrolling, and reconnaissance in high-risk environments. Additionally, the project highlights novel design approaches, such as efficient data mapping and optimized power management, ensuring robust performance in real-time scenarios. The report elaborates on the system's hardware integration, software algorithms, mechanical design, and power management.

System Design

Transmitter Section

Components Used:

- Arduino Nano
- NRF24L01+ Module
- NRF Adapter
- MPU6050 Module
- 7-12 V DC Battery (LIPO 2s Battery)
- Mini Breadboard
- Double Sided Tape
- Jumper Wires

Code Explanation:

The transmitter code is responsible for collecting motion data from the MPU6050 sensor, specifically the acceleration values for the X and Y axes. These raw values are adjusted by subtracting baseline calibration offsets and then mapped to a range of 0–254. The values are constrained to avoid exceeding hardware limits. The processed data is then packaged into a structure and sent wirelessly to the receiver using the nRF24L01 RF module. The code also includes a calibration routine to account for environmental and hardware variances by averaging multiple readings during setup.

Code Implementation:

```
#include <Wire.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <MPU6050.h>

RF24 radio(8, 9);
MPU6050 mpu;

const uint64_t pipeOut = 0xF9E8F0F0E1LL;

struct PacketData {
  byte xAxisValue;
  byte yAxisValue;
};

PacketData packet;

int16_t baseAx, baseAy, baseAz;
int16_t ax, ay, az, gx, gy, gz;

void setup() {
  Serial.begin(115200);
  radio.begin();
  radio.setDataRate(RF24_250KBPS);
  radio.openWritingPipe(pipeOut);
  Wire.begin();
  mpu.initialize();
  if (!mpu.testConnection()) {
```

```
    Serial.println("MPU6050 connection failed!");
    while (1);
  }
  calibrateMPU();
}

void loop() {
  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
  int mappedX = map(ax - baseAx, -17000, 17000, 0, 254);
  int mappedY = map(ay - baseAy, -17000, 17000, 0, 254);
  packet.xAxisValue = constrain(mappedX, 0, 254);
  packet.yAxisValue = constrain(mappedY, 0, 254);
  radio.write(&packet, sizeof(PacketData));
  Serial.print("X: ");
  Serial.print(packet.xAxisValue);
  Serial.print("\tY: ");
  Serial.println(packet.yAxisValue);
  delay(10);
}

void calibrateMPU() {
  int numSamples = 100;
  long sumAx = 0, sumAy = 0, sumAz = 0;

  for (int i = 0; i < numSamples; i++) {
    mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
    sumAx += ax;
    sumAy += ay;
    sumAz += az;
    delay(10);
  }

  baseAx = sumAx / numSamples;
  baseAy = sumAy / numSamples;
  baseAz = sumAz / numSamples;

  Serial.println("Calibration complete.");
}
```

Receiver Section

Components Used:

- 4 Wheels
- 4 DC Gear Motors
- Car Chasi
- Arduino Nano
- NRF24L01+ Module
- NRF Adapter
- L298N Driver Module
- 7-12 V DC battery (LIPO 3s Battery)
- Mini Breadboard

Code Explanation:

The receiver code listens for data packets from the transmitter via the nRF24L01 RF module. The received data contains the X and Y axis values, which represent the desired motion direction and intensity. These values are mapped to motor speed ranges (-255 to 255) for differential motor control. The robot's movement is governed by the motor speeds: forward, backward, or turning. A timeout mechanism ensures the robot stops if no signal is received within a specified duration, enhancing safety. The rotateMotor function dynamically adjusts motor speed and direction based on the processed input values.

Code Implementation:

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

#define SIGNAL_TIMEOUT 500

const uint64_t pipeIn = 0xF9E8F0F0E1LL;
RF24 radio(8, 9);

unsigned long lastRecvTime = 0;

struct PacketData {
  byte xAxisValue;
  byte yAxisValue;
} receiverData;
```

```
int enableRightMotor = 5;
int rightMotorPin1 = 2;
int rightMotorPin2 = 3;
int enableLeftMotor = 6;
int leftMotorPin1 = 4;
int leftMotorPin2 = 7;

void setup() {
  pinMode(enableRightMotor, OUTPUT);
  pinMode(rightMotorPin1, OUTPUT);
  pinMode(rightMotorPin2, OUTPUT);
  pinMode(enableLeftMotor, OUTPUT);
  pinMode(leftMotorPin1, OUTPUT);
  pinMode(leftMotorPin2, OUTPUT);
  rotateMotor(0, 0);
  radio.begin();
  radio.setDataRate(RF24_250KBPS);
  radio.openReadingPipe(1, pipeIn);
  radio.startListening();
}

void loop() {
  int rightMotorSpeed = 0;
  int leftMotorSpeed = 0;

  if (radio.isChipConnected() && radio.available()) {
    radio.read(&receiverData, sizeof(PacketData));
    int mappedYValue = map(receiverData.yAxisValue, 0, 254, -255, 255);
    int mappedXValue = map(receiverData.xAxisValue, 0, 254, -255, 255);
    rightMotorSpeed = mappedYValue - mappedXValue;
    leftMotorSpeed = mappedYValue + mappedXValue;
    rightMotorSpeed = constrain(rightMotorSpeed, -255, 255);
    leftMotorSpeed = constrain(leftMotorSpeed, -255, 255);
    rotateMotor(rightMotorSpeed, leftMotorSpeed);
    lastRecvTime = millis();
  } else {
    unsigned long now = millis();
    if (now - lastRecvTime > SIGNAL_TIMEOUT) {
      rotateMotor(0, 0);
    }
```

```
  }
}

void rotateMotor(int rightMotorSpeed, int leftMotorSpeed) {
  if (rightMotorSpeed < 0) {
    digitalWrite(rightMotorPin1, LOW);
    digitalWrite(rightMotorPin2, HIGH);
  } else if (rightMotorSpeed > 0) {
    digitalWrite(rightMotorPin1, HIGH);
    digitalWrite(rightMotorPin2, LOW);
  } else {
    digitalWrite(rightMotorPin1, LOW);
    digitalWrite(rightMotorPin2, LOW);
  }
  if (leftMotorSpeed < 0) {
    digitalWrite(leftMotorPin1, LOW);
    digitalWrite(leftMotorPin2, HIGH);
  } else if (leftMotorSpeed > 0) {
    digitalWrite(leftMotorPin1, HIGH);
    digitalWrite(leftMotorPin2, LOW);
  } else {
    digitalWrite(leftMotorPin1, LOW);
    digitalWrite(leftMotorPin2, LOW);
  }
  analogWrite(enableRightMotor, abs(rightMotorSpeed));
  analogWrite(enableLeftMotor, abs(leftMotorSpeed));
}
```

Mechanical Design

The robot chassis was carefully crafted to house the components securely while maintaining balance and stability. Custom soldered connections were meticulously designed to ensure efficient power flow, with particular attention to minimizing signal loss and electromagnetic interference. Creating reliable connections between the RF module, MPU6050 sensor, and motor driver required extensive troubleshooting and iterative testing. Additionally, managing power distribution for multiple components posed challenges in achieving consistent performance throughout the build process.

Electronic and Control System Design

In this section, we focus on the integration of electronic components and the control system that drives the robot based on gesture inputs. The robot utilizes an accelerometer and gyroscope sensor setup to detect and interpret the operator's hand movements. This sensor data is processed by an Arduino Nano microcontroller, which then controls the movement of the robot in real-time.

Sensor Integration

The accelerometer and gyroscope, often integrated into a single sensor module like the MPU6050, are strategically placed to capture the tilt, orientation, and movement of the operator's hand. These sensors measure the angular velocity and acceleration, providing the necessary data to control the robot's motion. The sensor outputs are sent to the Arduino, where the data is interpreted and mapped to control the robot's motors.

Control Algorithm

The core of the system is the control algorithm, which interprets the gesture signals from the sensors. The algorithm is designed to convert raw sensor data into motor control signals that steer the robot's movement. For instance:

- Forward motion is triggered by tilting the hand forward.
- Reverse motion is triggered by tilting the hand backward.
- Left or right turns are controlled by the rotational movement of the wrist.

The algorithm processes the sensor data and translates it into PWM (Pulse Width Modulation) signals that drive the motors controlling the wheels.

Communication System

The transmitter's arduino microcontroller communicates with the robot's drivers using radio frequencies. Wireless communication is incorporated, allowing the operator to control the robot from a greater distance, upto **800 meters**, which would be particularly beneficial for military applications where long-range control is essential.

Testing and Calibration

Before deployment, extensive testing and calibration are performed to ensure the accuracy and responsiveness of the gesture control system. The sensors are calibrated to account for environmental factors such as temperature and electromagnetic interference, which could impact their performance. Multiple test scenarios are conducted, where the robot's movement is evaluated under various conditions to fine-tune the control algorithm and sensor sensitivity. This ensures the robot's behavior is predictable, and any inconsistencies in sensor data are minimized for optimal performance in military applications.

Challenges Faced

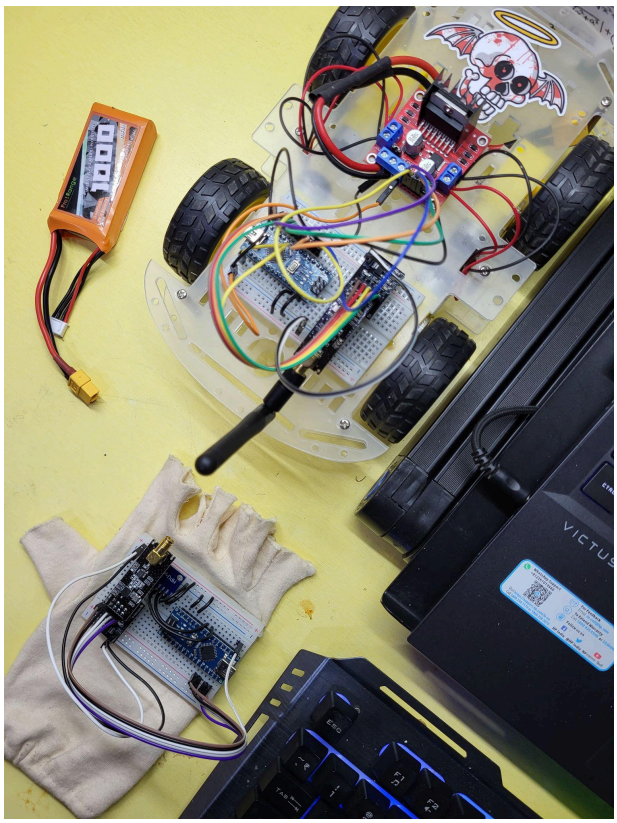
1. **Power Supply**
    - Calculating and delivering sufficient power to support motors, sensors, and communication modules.
    - Crafting custom wiring solutions for stable and efficient power distribution.

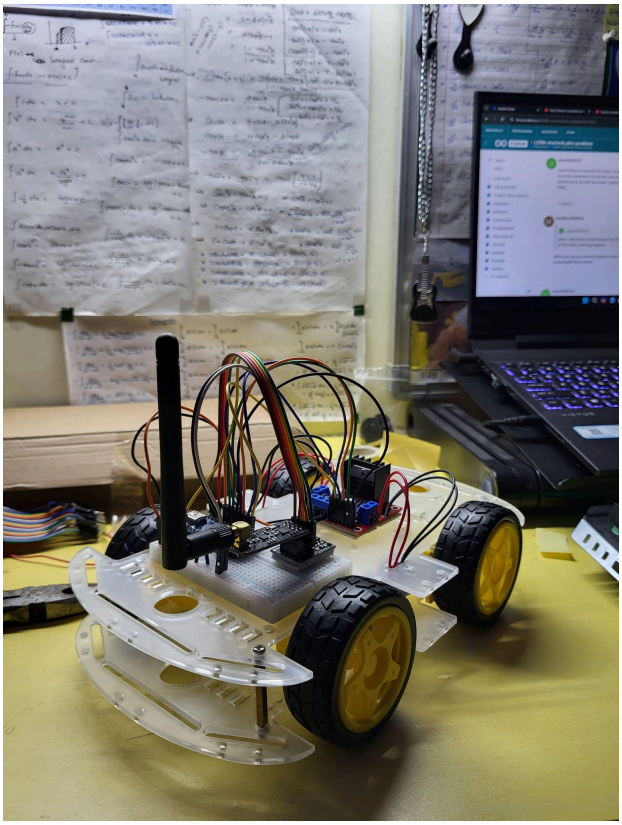
2. **Signal Noise**
    - Ensuring reliable RF communication in a noisy environment required optimizing transmitter-receiver settings.

3. **Mechanical Design**
    - Assembling stable components and soldering connections to withstand movement and vibrations.

4. **Coding Complexity**
    - Writing modular and efficient code for real-time gesture interpretation and robot movement synchronization.

A Few Clicks Of The Model:





Conclusion

This project provided valuable insights into integrating hardware and software for robotics applications. Overcoming mechanical and coding challenges underscored the importance of teamwork and iterative design in successful project execution. This gesture-controlled robot demonstrates potential in military applications, offering a safer alternative for tasks such as bomb defusal and border patrolling, paving the way for advancements in autonomous and semi-autonomous robotics.
